## Credit Card Analysis

**Team Members:**

Swaran Paramesh Kumar S (E0119015)

Santosh Prasad D (E0119050)

**Data Description :** The data set for this classification problem is taken from Kaggle.Dataset contains 25000 rows and 6 columns.

**Software Requirements and Platforms used:**

Python3 - for developing the model and implementation Google Colab -

Environment used to execute the Python code.

## Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

## Reading the dataset

```
Emp_Data = pd.read_excel('creditcard.xlsx')
Emp_Data
```

```
       Gender  Age     Occupation  Vintage  Avg_Account_Balance
Is_Lead
0      Female   80          Other       21               493496
0
1      Female   54  Self_Employed       20               908494
1
2        Male   46  Self_Employed       27               473391
1
3        Male   69          Other      105               863219
1
4        Male   29       Salaried       13               500785
0
...       ...  ...            ...      ...                  ...       .
..
24429    Male   81          Other       73              1516408
1
24430  Female   27          Other       15              1563302
0
24431    Male   56  Self_Employed       39               433750
0
24432  Female   57  Self_Employed       61               480422
0
```

```
24433    Male   30  Self_Employed       33              2229806
0

[24434 rows x 6 columns]
```

### ****Converting** categorical columns to numerical**

```
feats = ['Gender','Occupation']
emp_data_final =
pd.get_dummies(Emp_Data,columns=feats,drop_first=True)

emp_data_final

       Age  Vintage  ...  Occupation_Salaried
Occupation_Self_Employed
0       80       21  ...                    0
0
1       54       20  ...                    0
1
2       46       27  ...                    0
1
3       69      105  ...                    0
0
4       29       13  ...                    1
0
...    ...      ...  ...                  ...                        ..
.
24429   81       73  ...                    0
0
24430   27       15  ...                    0
0
24431   56       39  ...                    0
1
24432   57       61  ...                    0
1
24433   30       33  ...                    0
1

[24434 rows x 8 columns]
```

## CO1: Selection of Base Learners

### Splitting the dataset into features and target variable

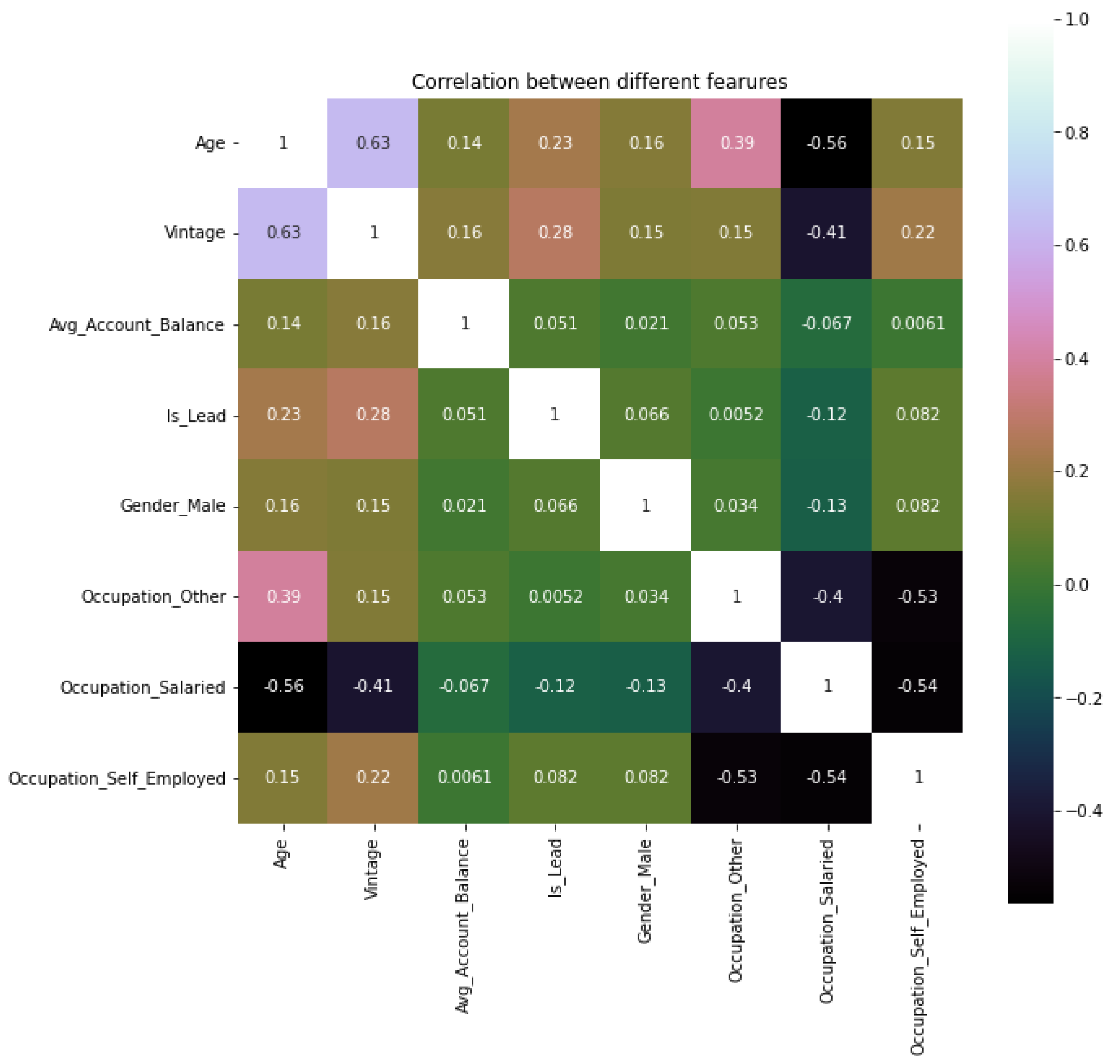
```
X = emp_data_final.drop(['Is_Lead'],axis=1).values
y = emp_data_final['Is_Lead'].values

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.3)
```

## Transforming the data to scale it

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
correlation = emp_data_final.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1,
square=True,annot=True,cmap='cubehelix')

plt.title('Correlation between different fearures')
```

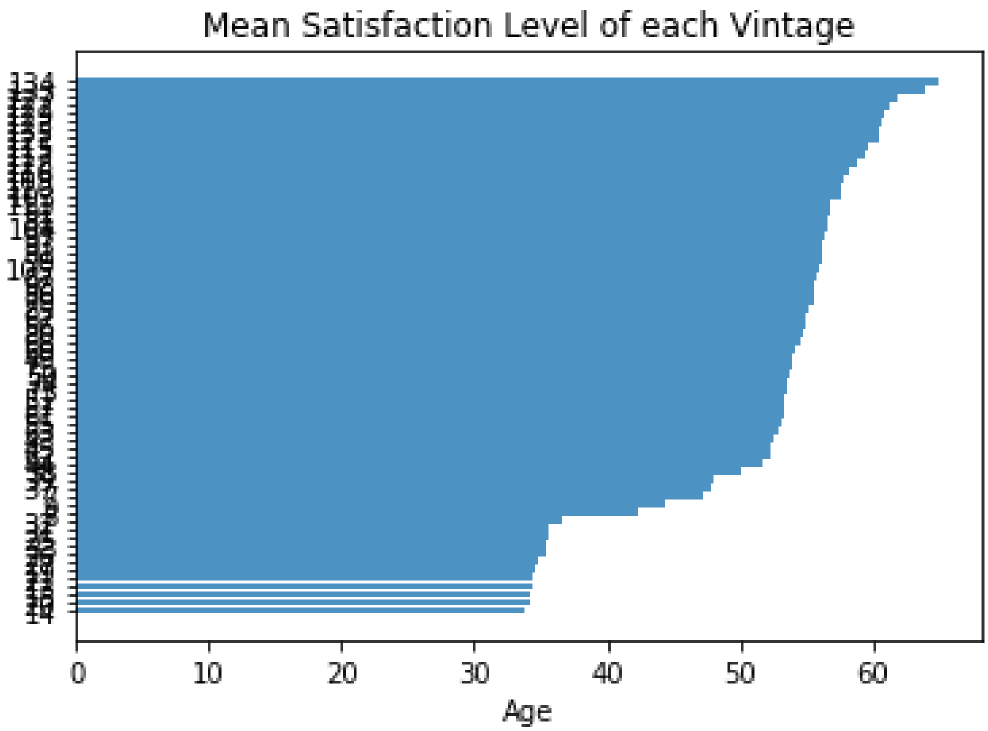
```
Text(0.5, 1.0, 'Correlation between different fearures')
```




```
satisfaction_by_dept=emp_data_final.groupby('Vintage').mean()
satisfaction_by_dept.sort_values(by="Age", ascending=True,
inplace=True)
```

```
satisfaction_by_dept
y_pos = np.arange(len(satisfaction_by_dept.index))

plt.barh(y_pos, satisfaction_by_dept['Age'], align='center',
alpha=0.8)
plt.yticks(y_pos, satisfaction_by_dept.index)

plt.xlabel('Age')
plt.title('Mean Satisfaction Level of each Vintage')
```

```
Text(0.5, 1.0, 'Mean Satisfaction Level of each Vintage')
```




## Voting is an ensemble ML algorithm.

### 1. HARD VOTING

Majority/ mode based voting algorithm

```
# Import required libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
#Hard Voting - We build our models with decision tree, support vector
machines and logistic regression algorithms
# Let's create the sub models
```

```
estimators = []

dt_model = DecisionTreeClassifier(random_state=1)
estimators.append(('DecisionTree', dt_model))

svm_model = SVC(random_state=1)
estimators.append(('SupportVector', svm_model))

logit_model = LogisticRegression(random_state=1)
estimators.append(('Logistic Regression', logit_model))

#We build individual models with each of the classifiers we have
chosen
from sklearn.metrics import accuracy_score

for each_estimator in (dt_model, svm_model, logit_model):
    each_estimator.fit(X_train, Y_train)
    Y_pred = each_estimator.predict(X_test)
    print(each_estimator.__class__.__name__, accuracy_score(Y_test,
Y_pred))

#We proceed to ensemble our models and use VotingClassifier to score
accuracy
# Using VotingClassifier() to build ensemble model with Hard Voting
ensemble_model = VotingClassifier(estimators=estimators,
voting='hard')
ensemble_model.fit(X_train,Y_train)
predicted_labels = ensemble_model.predict(X_test)
print("Classifier Accuracy using Hard Voting: ",
accuracy_score(Y_test, predicted_labels))
```

```
DecisionTreeClassifier 0.6932205701814214
SVC 0.7816123311962897
LogisticRegression 0.7567862501705088
Classifier Accuracy using Hard Voting:  0.7769744918837812
```

## 2.SOFT VOTING

It is the prediction of class based on the average of probability given to that class

```
#Soft Voting - The below code creates an ensemble using soft voting:
# create the sub models
estimators = []

dt_model = DecisionTreeClassifier(random_state=1)
estimators.append(('DecisionTree', dt_model))

svm_model = SVC(random_state=1, probability=True)
estimators.append(('SupportVector', svm_model))
```

```
logit_model = LogisticRegression(random_state=1)
estimators.append(('Logistic Regression', logit_model))

for each_estimator in (dt_model, svm_model, logit_model):
    each_estimator.fit(X_train, Y_train)
    Y_pred = each_estimator.predict(X_test)
    print(each_estimator.__class__.__name__, accuracy_score(Y_test,
Y_pred))
# Using VotingClassifier() to build ensemble model with Soft Voting
ensemble_model = VotingClassifier(estimators=estimators,
voting='soft')
ensemble_model.fit(X_train,Y_train)
predicted_labels = ensemble_model.predict(X_test)
print("Classifier Accuracy using Soft Voting: ",
accuracy_score(Y_test, predicted_labels))
```

```
DecisionTreeClassifier 0.6932205701814214
SVC 0.7816123311962897
LogisticRegression 0.7567862501705088
Classifier Accuracy using Soft Voting:  0.7614240894830173
```

## 3. Hyperparameter tuning ensemble

```
### Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf_1 = RandomForestClassifier(random_state=0, n_estimators=10)
rf_1.fit(X_train, Y_train)

rf_2 = RandomForestClassifier(random_state=0, n_estimators=50)
rf_2.fit(X_train, Y_train)

rf_3 = RandomForestClassifier(random_state=0, n_estimators=100)
rf_3.fit(X_train, Y_train)

# combine all three Voting Ensembles
from sklearn.ensemble import VotingClassifier
estimators = [('rf_1', rf_1), ('rf_2', rf_2), ('rf_3', rf_3)]
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(X_train, Y_train)
print("rf_1.score: ", rf_1.score(X_test, Y_test))
print("rf_2.score: ", rf_2.score(X_test, Y_test))
print("rf_3.score: ", rf_3.score(X_test, Y_test))
print("ensemble.score based on hard voting: ", ensemble.score(X_test,
Y_test))

estimators = [('rf_1', rf_1), ('rf_2', rf_2), ('rf_3', rf_3)]
ensemble = VotingClassifier(estimators, voting='soft')
ensemble.fit(X_train, Y_train)
print("rf_1.score: ", rf_1.score(X_test, Y_test))
print("rf_2.score: ", rf_2.score(X_test, Y_test))
print("rf_3.score: ", rf_3.score(X_test, Y_test))
```

```
print("ensemble.score based on soft voting: ", ensemble.score(X_test, Y_test))
```

```
rf_1.score:  0.7494202700859365
rf_2.score:  0.7563770290546992
rf_3.score:  0.7550129586686674
ensemble.score based on hard voting:  0.7565134360933025
rf_1.score:  0.7494202700859365
rf_2.score:  0.7563770290546992
rf_3.score:  0.7550129586686674
ensemble.score based on soft voting:  0.7532396671668258
```

## CO2:Ensemble Learning - Meta heuristics

### 4. Bagging

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
bag = BaggingClassifier(base_estimator=SVC(),
n_estimators=10, random_state=0).fit(X_train, Y_train)
print(bag.score(X_test, Y_test))
```

```
0.7813395171190833
```

### 5. Stacking

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier

#base learners
estimators = [
              ("rf", RandomForestClassifier(n_estimators=100,
random_state=42)),
              ("svr", make_pipeline(StandardScaler(),
LinearSVC(max_iter=10000,random_state=42))),
              ]
clf = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())

clf.fit(X_train, Y_train).score(X_test, Y_test)
```

```
0.7757468285363525
```

## 6. Boosting

### AdaBoost

```
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(n_estimators=100).fit(X_train,Y_train)
scores = cross_val_score(clf, X_test, Y_test, cv=5)
print(scores.mean())
```

```
0.7722006005704396
```

### GradientBoosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0).fit(X_train,Y_train)
scores = cross_val_score(clf, X_test, Y_test, cv=5)
print(scores.mean())
```

```
0.7749292995235797
```

### XGBoosting

```
import warnings
warnings.filterwarnings('ignore')
import xgboost as xgb
dtrain = xgb.DMatrix(X_train, label=Y_train)
dtest = xgb.DMatrix(X_test, label=Y_test)
# set xgboost params
param = {
'max_depth': 5, # the maximum depth of each tree
'eta': 0.3, # the training step for each iteration
'silent': 1, # logging mode - quiet
'objective': 'multi:softprob', # error evaluation for multiclass
training
'num_class': 3} # the number of classes that exist in this datset
num_round = 200 # the number of training iterations

bst = xgb.train(param, dtrain, num_round)

# make prediction
preds = bst.predict(dtest)
preds_rounded = np.argmax(preds, axis=1)
print(accuracy_score(Y_test, preds_rounded))
```

```
0.7720638384940663
```

## CO3: Feature Selection and Report on Hyper parameters

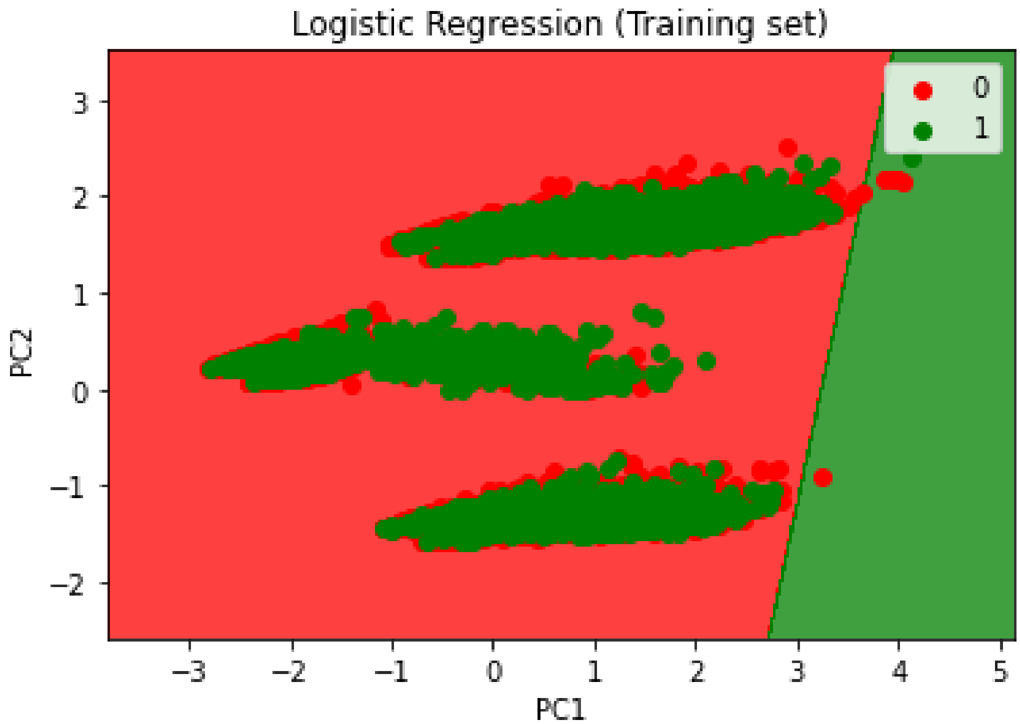
### Plotting Decision regions

### 7.Principal Component Analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train_pca, Y_train)

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test_pca)
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)

from matplotlib.colors import ListedColormap
X_set, y_set = X_train_pca, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = 
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = 
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), 
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

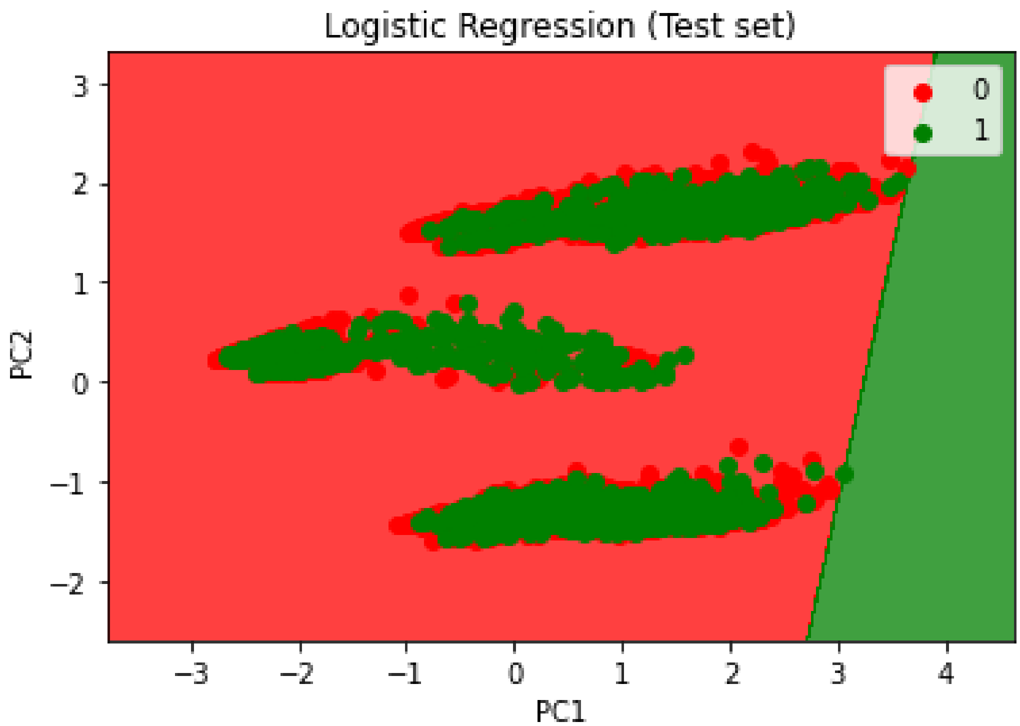
```
[[5592    0]
 [1739    0]]
```




```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test_pca, Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```



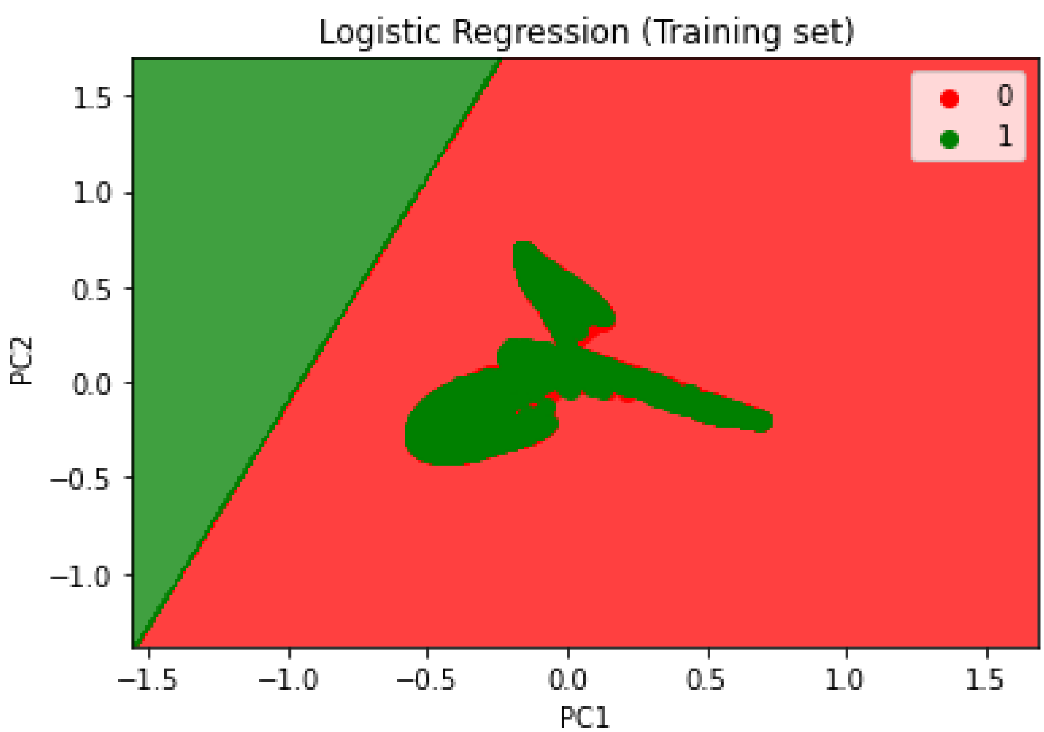

### Kernel PCA

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 2, kernel = 'rbf')
X_train_kpca = kpca.fit_transform(X_train)
X_test_kpca = kpca.transform(X_test)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train_kpca, Y_train)

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test_kpca)
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)

from matplotlib.colors import ListedColormap
X_set, y_set = X_train_kpca, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
```

```
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

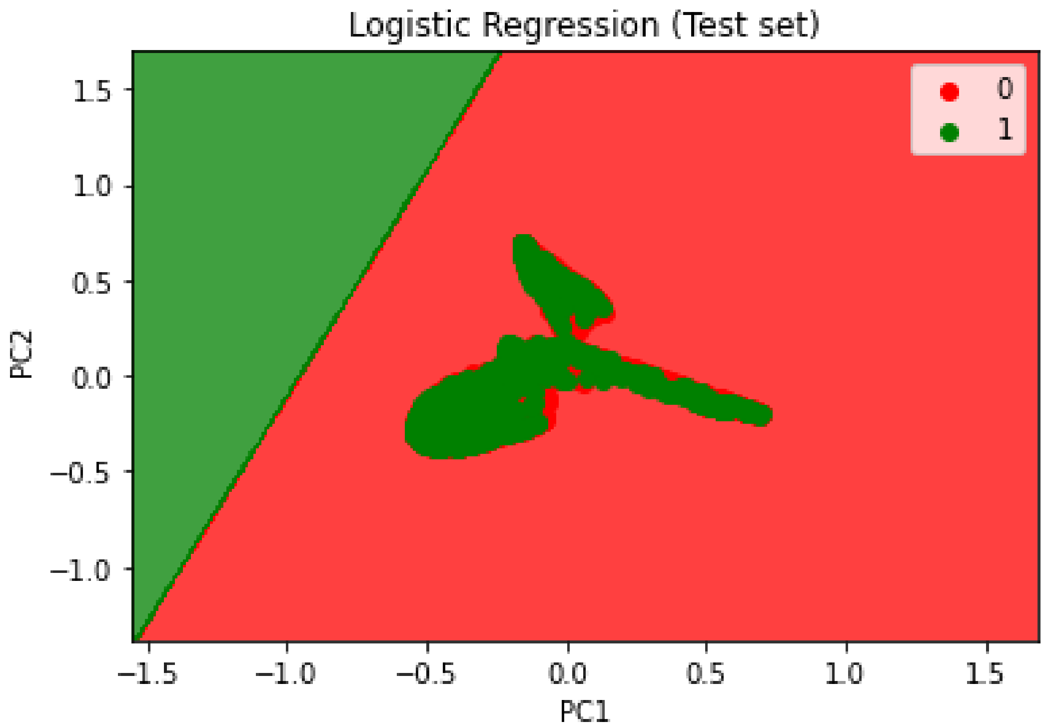
```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

```
[[5592    0]
 [1739    0]]
```



```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test_kpca, Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

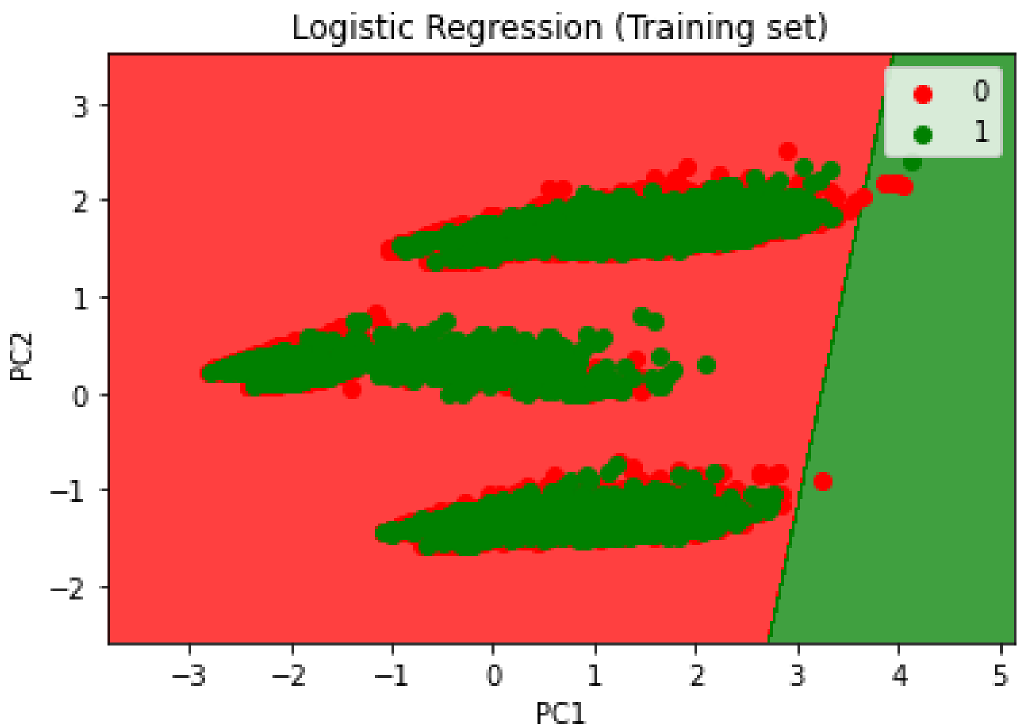



### 9. Linear Discriminant Analysis

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train_pca, Y_train)

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test_pca)
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)

from matplotlib.colors import ListedColormap
X_set, y_set = X_train_pca, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')
```

```
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which
should be avoided as value-mapping will have precedence in case its
length matches with *x* & *y*.  Please use the *color* keyword-
argument or provide a 2-D array with a single row if you intend to
specify the same RGB or RGBA value for all points.

[[5592    0]
 [1739    0]]
```




## 10.Perceptron

### Evaluating a perceptron model on the dataset

```
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define model
model = Perceptron()
```

```
# define model evaluation method : Cross Validation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, Y_train, scoring='accuracy',
cv=cv, n_jobs=-1)
# summarize result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean Accuracy: 0.677 (0.059)
```

### Making a prediction with the perceptron model on the dataset

```
model.fit(X_train, Y_train)
# define new data
row = [0.41,0.57,8,200,3,1,1]
# make a prediction
yhat = model.predict([row])
# summarize prediction
print('Predicted Class: %d' % yhat)
```

```
Predicted Class: 1
```

### Grid search learning rate for the perceptron

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['eta0'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-
1)
# perform the search
results = search.fit(X_train, Y_train)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
# summarize all
means = results.cv_results_['mean_test_score']
params = results.cv_results_['params']
for mean, param in zip(means, params):
    print(">%.3f with: %r" % (mean, param))
```

```
Mean Accuracy: 0.678
Config: {'eta0': 0.1}
>0.676 with: {'eta0': 0.0001}
>0.676 with: {'eta0': 0.001}
>0.676 with: {'eta0': 0.01}
>0.678 with: {'eta0': 0.1}
>0.677 with: {'eta0': 1.0}
```

### Grid search total epochs for the perceptron

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define model
model = Perceptron(eta0=0.0001)
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['max_iter'] = [1, 10, 100, 1000, 10000]
# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-
1)
# perform the search
results = search.fit(X_train, Y_train)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
# summarize all
means = results.cv_results_['mean_test_score']
params = results.cv_results_['params']
for mean, param in zip(means, params):
    print(">%.3f with: %r" % (mean, param))
```

```
Mean Accuracy: 0.676
Config: {'max_iter': 10}
>0.667 with: {'max_iter': 1}
>0.676 with: {'max_iter': 10}
>0.676 with: {'max_iter': 100}
>0.676 with: {'max_iter': 1000}
>0.676 with: {'max_iter': 10000}
```

# CO4: Building ANN Models

## 11.Artificial Neural Network

```
import tensorflow as tf
from tensorflow import keras

ann = tf.keras.models.Sequential()
# Adding the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```python
# Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics
= ['accuracy'])

# Training the ANN on the Training set
ann.fit(X_train, Y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
535/535 [==============================] - 2s 2ms/step - loss: 0.5375
- accuracy: 0.7536
Epoch 2/100
535/535 [==============================] - 1s 2ms/step - loss: 0.5089
- accuracy: 0.7581
Epoch 3/100
535/535 [==============================] - 1s 2ms/step - loss: 0.5008
- accuracy: 0.7660
Epoch 4/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4932
- accuracy: 0.7757
Epoch 5/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4887
- accuracy: 0.7797
Epoch 6/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4865
- accuracy: 0.7797
Epoch 7/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4848
- accuracy: 0.7807
Epoch 8/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4839
- accuracy: 0.7817
Epoch 9/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4829
- accuracy: 0.7814
Epoch 10/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4822
- accuracy: 0.7822
Epoch 11/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4816
- accuracy: 0.7823
Epoch 12/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4809
- accuracy: 0.7821
Epoch 13/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4804
- accuracy: 0.7814
Epoch 14/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4797
- accuracy: 0.7819
Epoch 15/100
```

```
535/535 [==============================] - 1s 2ms/step - loss: 0.4793
- accuracy: 0.7825
Epoch 16/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4790
- accuracy: 0.7817
Epoch 17/100
535/535 [==============================] - 1s 1ms/step - loss: 0.4787
- accuracy: 0.7820
Epoch 18/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4782
- accuracy: 0.7823
Epoch 19/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4778
- accuracy: 0.7825
Epoch 20/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4775
- accuracy: 0.7818
Epoch 21/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4771
- accuracy: 0.7827
Epoch 22/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4769
- accuracy: 0.7817
Epoch 23/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4767
- accuracy: 0.7824
Epoch 24/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4763
- accuracy: 0.7828
Epoch 25/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4761
- accuracy: 0.7821
Epoch 26/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4760
- accuracy: 0.7821
Epoch 27/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4756
- accuracy: 0.7824
Epoch 28/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4756
- accuracy: 0.7823
Epoch 29/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4752
- accuracy: 0.7820
Epoch 30/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4749
- accuracy: 0.7826
Epoch 31/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4751
- accuracy: 0.7823
```

```
Epoch 32/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4745
- accuracy: 0.7830
Epoch 33/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4741
- accuracy: 0.7825
Epoch 34/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4738
- accuracy: 0.7823
Epoch 35/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4734
- accuracy: 0.7833
Epoch 36/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4732
- accuracy: 0.7824
Epoch 37/100
535/535 [==============================] - 1s 1ms/step - loss: 0.4726
- accuracy: 0.7830
Epoch 38/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4724
- accuracy: 0.7828
Epoch 39/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4722
- accuracy: 0.7837
Epoch 40/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4718
- accuracy: 0.7834
Epoch 41/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4711
- accuracy: 0.7841
Epoch 42/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4711
- accuracy: 0.7836
Epoch 43/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4706
- accuracy: 0.7834
Epoch 44/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4702
- accuracy: 0.7838
Epoch 45/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4700
- accuracy: 0.7840
Epoch 46/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4697
- accuracy: 0.7840
Epoch 47/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4692
- accuracy: 0.7833
Epoch 48/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4689
```

```
- accuracy: 0.7828
Epoch 49/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4687
- accuracy: 0.7834
Epoch 50/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4683
- accuracy: 0.7841
Epoch 51/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4678
- accuracy: 0.7841
Epoch 52/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4677
- accuracy: 0.7841
Epoch 53/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4675
- accuracy: 0.7840
Epoch 54/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4673
- accuracy: 0.7839
Epoch 55/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4671
- accuracy: 0.7848
Epoch 56/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4669
- accuracy: 0.7842
Epoch 57/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4669
- accuracy: 0.7844
Epoch 58/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4666
- accuracy: 0.7839
Epoch 59/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4664
- accuracy: 0.7839
Epoch 60/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4663
- accuracy: 0.7837
Epoch 61/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4660
- accuracy: 0.7835
Epoch 62/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4661
- accuracy: 0.7839
Epoch 63/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4661
- accuracy: 0.7838
Epoch 64/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4659
- accuracy: 0.7843
Epoch 65/100
```

```
535/535 [==============================] - 1s 2ms/step - loss: 0.4658
- accuracy: 0.7832
Epoch 66/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4654
- accuracy: 0.7834
Epoch 67/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4654
- accuracy: 0.7833
Epoch 68/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4654
- accuracy: 0.7842
Epoch 69/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4655
- accuracy: 0.7831
Epoch 70/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4652
- accuracy: 0.7835
Epoch 71/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4651
- accuracy: 0.7838
Epoch 72/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4650
- accuracy: 0.7840
Epoch 73/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4650
- accuracy: 0.7828
Epoch 74/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4651
- accuracy: 0.7837
Epoch 75/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4650
- accuracy: 0.7839
Epoch 76/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4651
- accuracy: 0.7834
Epoch 77/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4647
- accuracy: 0.7831
Epoch 78/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4646
- accuracy: 0.7828
Epoch 79/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4647
- accuracy: 0.7829
Epoch 80/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4646
- accuracy: 0.7837
Epoch 81/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4646
- accuracy: 0.7840
```

```
Epoch 82/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4646
- accuracy: 0.7849
Epoch 83/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4643
- accuracy: 0.7845
Epoch 84/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4645
- accuracy: 0.7840
Epoch 85/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4636
- accuracy: 0.7842
Epoch 86/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4641
- accuracy: 0.7835
Epoch 87/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4639
- accuracy: 0.7835
Epoch 88/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4643
- accuracy: 0.7839
Epoch 89/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4638
- accuracy: 0.7840
Epoch 90/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4637
- accuracy: 0.7842
Epoch 91/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4637
- accuracy: 0.7839
Epoch 92/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4633
- accuracy: 0.7841
Epoch 93/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4634
- accuracy: 0.7841
Epoch 94/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4632
- accuracy: 0.7838
Epoch 95/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4635
- accuracy: 0.7843
Epoch 96/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4634
- accuracy: 0.7842
Epoch 97/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4632
- accuracy: 0.7854
Epoch 98/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4630
```

```
- accuracy: 0.7848
Epoch 99/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4631
- accuracy: 0.7842
Epoch 100/100
535/535 [==============================] - 1s 2ms/step - loss: 0.4630
- accuracy: 0.7851

<keras.callbacks.History at 0x7fba74e10390>
```

### Making a prediction with the ANN model on the dataset

```
print(ann.predict(sc.transform([[0.41,0.57,8,200,3,1,1]])) > 0.5)

[[False]]
```

### Predicting and Evaluating the ANN model

```
# Predicting the Test set results
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
Y_test.reshape(len(Y_test),1)),1))

[[0 0]
 [0 0]
 [0 0]
 ...
 [0 1]
 [0 0]
 [0 0]]

#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)

[[5506  115]
 [1454  256]]

0.7859773564315918
```

## 12.Homogeneous ensemble

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras import optimizers

df_traindata, df_testdata = train_test_split(emp_data_final,
test_size=0.3)

print(df_traindata.shape)
print(df_testdata.shape)
```

```
(17103, 8)
(7331, 8)
```

```
X_test = df_testdata.drop(['Is_Lead'],axis=1).values
Y_test = df_testdata['Is_Lead'].values

print(X_test.shape)
print(Y_test.shape)
```

```
(7331, 7)
(7331,)
```

### Using keras to build NN model

```
learning_rate=0.001
ensemble = 3
frac = 0.7

predictions_total = np.zeros(7331, dtype=float)

for i in range(ensemble):
  print("number of iterations:", i)
  print("predictions_total",predictions_total)

  #sample randomly the train data
  traindata = df_traindata.sample(frac=frac)
  X_train = traindata.drop(['Is_Lead'],axis=1).values
  Y_train = traindata['Is_Lead'].values

  model = Sequential()
  #Adding the input layer and First hidden layer
  model.add(Dense(units=36, kernel_initializer='normal',
activation='relu', input_dim= 7))

  #Add Second hidden layer
  model.add(Dense(units=24, kernel_initializer='normal',
activation='relu'))

  #Add Third hidden layer
  model.add(Dense(units=16, kernel_initializer='normal',
activation='relu'))

  #Add output layer
  model.add(Dense(units=1, kernel_initializer='normal',
activation='relu'))

  # Compiling the ANN
  adam = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9,
beta_2=0.999, epsilon=None, decay=0.0)
  model.compile(loss='mse', optimizer=adam,
metrics=['mean_squared_error'])
```

```
  model.fit(X_train, Y_train, batch_size=32, epochs=100)

  model_predictions = model.predict(X_test)
  model_predictions = model_predictions.flatten()

  print("TEST MSE for individual Models",mean_squared_error(Y_test,
model_predictions))
  print("")
  print(model_predictions)
  print("")

  predictions_total = np.add(predictions_total, model_predictions)
```

```
number of iterations: 0
predictions_total [0. 0. 0. ... 0. 0. 0.]
Epoch 1/100
375/375 [==============================] - 1s 2ms/step - loss:
3672.5071 - mean_squared_error: 3672.5071
Epoch 2/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 3/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 4/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 5/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 6/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 7/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 8/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 9/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 10/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 11/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 12/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 13/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 14/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 15/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 16/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 17/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 18/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 19/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 20/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 21/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 22/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 23/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 24/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 25/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 26/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 27/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 28/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
```

```
Epoch 29/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 30/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 31/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 32/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 33/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 34/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 35/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 36/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 37/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 38/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 39/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 40/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 41/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 42/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 43/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 44/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 45/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
```

```
- mean_squared_error: 0.2379
Epoch 46/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 47/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 48/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 49/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 50/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 51/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 52/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 53/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 54/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 55/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 56/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 57/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 58/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 59/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 60/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 61/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 62/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 63/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 64/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 65/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 66/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 67/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 68/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 69/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 70/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 71/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 72/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 73/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 74/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 75/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 76/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 77/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 78/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
```

```
Epoch 79/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 80/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 81/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 82/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 83/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 84/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 85/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 86/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 87/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 88/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 89/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 90/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 91/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 92/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 93/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 94/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 95/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
```

```
- mean_squared_error: 0.2379
Epoch 96/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 97/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 98/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 99/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
Epoch 100/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2379
- mean_squared_error: 0.2379
TEST MSE for individual Models 0.23789387532396672

[0. 0. 0. ... 0. 0. 0.]

number of iterations: 1
predictions_total [0. 0. 0. ... 0. 0. 0.]
Epoch 1/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 2/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 3/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 4/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 5/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 6/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 7/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 8/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 9/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 10/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 11/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 12/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 13/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 14/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 15/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 16/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 17/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 18/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 19/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 20/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 21/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 22/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 23/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 24/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 25/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 26/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
```

```
Epoch 27/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 28/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 29/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 30/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 31/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 32/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 33/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 34/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 35/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 36/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 37/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 38/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 39/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 40/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 41/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 42/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 43/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
```

```
- mean_squared_error: 0.2363
Epoch 44/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 45/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 46/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 47/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 48/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 49/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 50/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 51/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 52/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 53/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 54/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 55/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 56/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 57/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 58/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 59/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 60/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 61/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 62/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 63/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 64/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 65/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 66/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 67/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 68/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 69/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 70/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 71/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 72/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 73/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 74/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 75/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 76/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
```

```
Epoch 77/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 78/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 79/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 80/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 81/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 82/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 83/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 84/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 85/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 86/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 87/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 88/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 89/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 90/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 91/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 92/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 93/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
```

```
- mean_squared_error: 0.2363
Epoch 94/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 95/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 96/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 97/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 98/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 99/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
Epoch 100/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2363
- mean_squared_error: 0.2363
TEST MSE for individual Models 0.23789387532396672

[0. 0. 0. ... 0. 0. 0.]

number of iterations: 2
predictions_total [0. 0. 0. ... 0. 0. 0.]
Epoch 1/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 2/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 3/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 4/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 5/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 6/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 7/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 8/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 9/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 10/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 11/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 12/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 13/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 14/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 15/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 16/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 17/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 18/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 19/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 20/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 21/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 22/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 23/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 24/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
```

```
Epoch 25/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 26/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 27/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 28/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 29/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 30/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 31/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 32/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 33/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 34/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 35/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 36/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 37/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 38/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 39/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 40/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 41/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
```

```
- mean_squared_error: 0.2399
Epoch 42/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 43/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 44/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 45/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 46/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 47/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 48/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 49/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 50/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 51/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 52/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 53/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 54/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 55/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 56/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 57/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 58/100
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 59/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 60/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 61/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 62/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 63/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 64/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 65/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 66/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 67/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 68/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 69/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 70/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 71/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 72/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 73/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 74/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
```

```
Epoch 75/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 76/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 77/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 78/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 79/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 80/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 81/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 82/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 83/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 84/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 85/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 86/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 87/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 88/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 89/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 90/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 91/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
```

```
- mean_squared_error: 0.2399
Epoch 92/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 93/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 94/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 95/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 96/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 97/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 98/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 99/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
Epoch 100/100
375/375 [==============================] - 1s 2ms/step - loss: 0.2399
- mean_squared_error: 0.2399
TEST MSE for individual Models 0.23789387532396672

[0. 0. 0. ... 0. 0. 0.]
```

## CO5 : Comparsion of Performance

### Printing average of predictions

```
from sklearn.metrics import mean_squared_error
predictions_total = predictions_total/ensemble
print("MSE after ensemble:", mean_squared_error(np.array(Y_test),
predictions_total))
print("")
print(predictions_total)
```

```
MSE after ensemble: 0.23789387532396672

[0. 0. 0. ... 0. 0. 0.]
```

Comparision of Models

Classifier Accuracy using Hard Voting: 0.7769744918837812

Classifier Accuracy using Soft Voting: 0.7614240894830173

ensemble.score based on hard voting: 0.7565134360933025

ensemble.score based on soft voting: 0.7532396671668258

Bagging Acccuracy : 0.7813395171190833

Stacking: 0.7757468285363525

AdaBoost: 0.7722006005704396

GradientBoosting: 0.7749292995235797

XGBoosting: 0.7720638384940663

Perceptron Mean Accuracy: 0.677 (0.059)

Grid search learning rate for the perceptron:

Mean Accuracy: 0.678

Grid search total epochs for the perceptron

Mean Accuracy: 0.676

Predicting model with ANN

[[5506 115] [1454 256]] 0.7859773564315918